# 理論的かつ実用的に高速な厳密文字列照合アルゴリズムに関する研究

修士学位論文要約（令和２年３月）

# 理論的かつ実用的に高速な厳密文字列照合アルゴリズムに関する研究

小林 賢
指導教員：篠原 歩，　学位論文指導教員：吉仲 亮

# Theoretically and practically fast exact pattern matching algorithms

Satoshi KOBAYASHI
Supervisor: Ayumi SHINOHARA, Research Advisor: Ryo YOSHINAKA

Given a text $T$ and a pattern $P$, the exact pattern matching problem is a task to find all occurrences of $P$ in $T$. In this study, we propose algorithms that solve this problem. The first is a modified version of the Franek-Jennings-Smyth algorithm. The second is based on the distance between two adjacent occurrences of the same $q$-gram contained in $P$. We also propose a theoretical improvement of the second algorithm which runs in linear time, though it is not necessarily faster in practice. We compare the execution times of our and existing algorithms on various kinds of datasets such as an English text and a genome sequence. The experimental results show that the second algorithm is as fast as the state-of-the-art algorithms in many cases, particularly when a pattern frequently appears in a text.

## 1. はじめに

厳密文字列照合問題は，長さ $n$ のテキスト $T$ と長さ $m$ のパターン $P$ が与えられたときに，$T$ 中の $P$ と等しい部分文字列の出現位置を全て見つける問題であり，文字列処理において最も基本的な問題の一つである．この問題を解く素朴な解法は，$T$ 中の全ての長さ $m$ の部分文字列と $P$ を比較することであり，$O(nm)$ 時間を必要とする．KMP アルゴリズム [3] は，$P$ の周期性を利用することによってこの問題を $O(n+m)$ 時間で解くことができる．また，Sunday アルゴリズム [4] は，$P$ 中の各文字の出現位置を利用したアルゴリズムであり，照合の最悪時間計算量が $O(nm)$ であるにも関わらず実用的には KMP アルゴリズムよりも高速に動作する．しかし，このようなアルゴリズムをソフトウェアに組み込んで実際に使用することを考えた場合，最悪なケースの文字列が与えられたときにソフトウェアの動作が遅くなってしまう恐れがある．そこで本研究では，理論的かつ実用的に高速に動作するアルゴリズムをいくつか提案する．１つ目のアルゴリズムの最悪時間計算量は前処理が $O(m+\sigma)$，照合が $O(n)$ であり，２つ目は前処理が $O(mq)$，照合が $O(nq)$ である．ここで，$\sigma$ はアルファベットサイズである．また，後者のアルゴリズムの前処理と照合の最悪時間計算量をそれぞれ $O(m)$，$O(n)$ に削減したアルゴリズムも提案する．そして，実験によって多くのケースにおいて提案手法が既存の有力なアルゴリズムと同等もしくはそれ以上に高速に動作することを示す．

## 2. FJS アルゴリズムを改善したアルゴリズム（FJS+）

FJS アルゴリズム [2] は，KMP アルゴリズムと Sunday アルゴリズム [4] を組み合わせたアルゴリズムである．KMP アルゴリズムは線形時間で動作するが，実際はそれほど高速に動作しない．一方，Sunday アルゴリズムは，最悪時間計算量は良くないが実際には高速に動作するアルゴリズムである．FJS アルゴリズムは，KMP と Sunday アルゴリズム両方の長所を持っている．最悪時間計算量は，前処理が $O(m+\sigma)$ で照合は $O(n)$ である．そして，実際にも高速に動作する．提案手法では，FJS アルゴリズムに Quite-Naive アルゴリズム [1] のアイデアを組み合わせることで，最悪時間計算量はそのままに，パターンのシフト量を増加させる．

## 3. 同一 $q$ グラムの直近の出現位置間の距離を考慮したアルゴリズム（DIST$q$）

FJS アルゴリズムでは照合を高速化するために Sunday アルゴリズムのシフトを用いるが，アルファベットサイズが小さいときにシフト量が小さくなるため効率が落ちる．そのようなときの効率の低下を防ぐため，Sunday アルゴリズムのように１文字に注目するのではなく，$q$ グラムに注目することでパターンのシフト量を大きくする．本提案手法では，FJS アルゴリズムのように KMP アルゴリズムを組み合わせるテクニックを使いつつ，同一 $q$ グラムの直近の出現位置間の距離を考慮することで照合速度

を向上させる．実装の都合上，文字列のハッシュ値を用いて $q$ グラム同士の比較を行う．最悪時間計算量は，前処理は $m-q+1$ 箇所で $q$ グラムのハッシュ値の計算を行うため，$O(mq)$ であり，照合は最大 $n-q+1$ 箇所でハッシュ値の計算を行うため，$O(nq)$ である．

### 線形時間アルゴリズム（LDIST$q$）

DIST$q$ アルゴリズムで用いられるハッシュ関数がローリングハッシュである場合，ある文字列 $w$ について $w[i:j]$ のハッシュ値がすでに求まっているとき，$w[i+1:j+1]$ のハッシュ値を定数時間で計算することが可能である．そのため，$q$ グラムのハッシュ値の差分更新をうまく行うことで照合の計算量を $O(n)$ に削減することができる．同様に，前処理の計算量も $O(m)$ に削減することができる．

## 4. 実験

いくつかのデータセットを用いて提案手法と既存アルゴリズムの実行速度の比較を行う．実験に用いる全てのアルゴリズムは C 言語にて実装されている．コンパイラは，GCC 9.2.0 を使用し，`-O3` オプションを付けて最適化を行っている．実験環境は，MacBook Pro (13-inch，2018)，macOS Catalina，Intel Core i7 2.7GHz クアッドコア，16GB メモリである．次のデータセットを用いて実験を行った．

1. ゲノム配列（表 1）: 大腸菌のゲノム配列（$n=4641652$，$\sigma=4$）
2. 英文（表 2）: 欽定訳聖書（$n=4017009$，$\sigma=62$）
3. パターン出現数を調整した文字列（表 3）: パターンは，ランダムに生成した長さ $m=16$ の文字列である．テキストは，ランダムに生成した長さ $n=4000000$, $\sigma=8$ の文字列にパターンを重ならないように埋め込み，パターン出現数を調整したものである．

なお各結果は，最良の実行時間（ミリ秒）が記録されており，括弧の中に書かれた数字はその結果が得られた $q$ の値を示している．ゲノム配列と英文の実験では，提案手法の DIST$q$ アルゴリズムが既存の最速アルゴリズムとほぼ同等またはそれよりもわずかに高速に動作していることが分かる．パターン出現数を調整したランダム文字列における実験では，パターン出現数が少ないときには既存手法の方が高速であった．しかし，出現数が増えていくにつれて既存手法は動作速度が遅くなっていくのに対し，DIST$q$ アルゴリズムはそれほど効率が悪化していないことが確認できる．よって，DIST$q$ アルゴリズムは最先端の既存アルゴリズムと同等もしくはそれ以上に高速に動作し，特にパターンがテキストに多く出現する場合において効率的であると考えられる．

表 1．ゲノム配列

| $m$ | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|
| BNDM$q$ | 179.32$^{(2)}$ | 90.01$^{(4)}$ | 72.30$^{(4)}$ | 63.64$^{(6)}$ | 61.42$^{(6)}$ | 62.93$^{(6)}$ | 64.09$^{(6)}$ |
| SBNDM$q$ | 157.66$^{(2)}$ | **83.44$^{(4)}$** | **70.16$^{(4)}$** | **61.84$^{(6)}$** | 62.37$^{(6)}$ | 62.48$^{(6)}$ | 61.96$^{(6)}$ |
| KBNDM | 198.89 | 170.13 | 120.59 | 90.17 | 76.96 | 76.03 | 75.20 |
| BSDM$q$ | **125.89$^{(3)}$** | 83.68$^{(4)}$ | 71.87$^{(6)}$ | 63.09$^{(8)}$ | 59.84$^{(8)}$ | 58.50$^{(8)}$ | 58.64$^{(8)}$ |
| FJS | 344.21 | 298.26 | 275.94 | 286.87 | 288.44 | 267.46 | 309.48 |
| FJS+ | 284.58 | 210.11 | 167.76 | 153.53 | 143.64 | 127.02 | 132.49 |
| HASH$q$ | 271.41$^{(3)}$ | 132.54$^{(3)}$ | 94.60$^{(3)}$ | 75.16$^{(4)}$ | 65.45$^{(4)}$ | 62.67$^{(6)}$ | 60.20$^{(7)}$ |
| FS-$w$ | 243.11$^{(4)}$ | 188.45$^{(6)}$ | 161.70$^{(4)}$ | 153.63$^{(4)}$ | 132.98$^{(4)}$ | 124.72$^{(4)}$ | 119.84$^{(4)}$ |
| LWFR$q$ | 170.60$^{(3)}$ | 95.50$^{(4)}$ | 70.65$^{(5)}$ | 62.57$^{(5)}$ | **58.68$^{(7)}$** | **56.87$^{(7)}$** | **56.20$^{(3)}$** |
| DIST$q$ | 126.98$^{(3)}$ | 87.10$^{(4)}$ | 70.74$^{(5)}$ | 62.39$^{(6)}$ | 59.94$^{(6)}$ | 58.30$^{(7)}$ | 57.35$^{(6)}$ |
| LDIST$q$ | 190.05$^{(3)}$ | 93.85$^{(4)}$ | 75.96$^{(5)}$ | 65.60$^{(5)}$ | 60.18$^{(6)}$ | 59.05$^{(6)}$ | 57.78$^{(8)}$ |

表 2．英文

| $m$ | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|
| BNDM$q$ | 90.24$^{(2)}$ | 74.49$^{(4)}$ | 59.75$^{(4)}$ | **53.47$^{(4)}$** | 53.69$^{(4)}$ | 53.20$^{(4)}$ | 52.73$^{(4)}$ |
| SBNDM$q$ | **78.90$^{(2)}$** | 71.15$^{(2)}$ | 58.70$^{(4)}$ | 55.18$^{(6)}$ | 54.18$^{(4)}$ | 54.12$^{(6)}$ | 53.95$^{(6)}$ |
| KBNDM | 129.10 | 95.89 | 77.76 | 71.20 | 63.25 | 57.34 | 56.16 |
| BSDM$q$ | 81.33$^{(2)}$ | 70.78$^{(3)}$ | 58.48$^{(5)}$ | 53.78$^{(6)}$ | 51.18$^{(6)}$ | 50.32$^{(6)}$ | 50.12$^{(6)}$ |
| FJS | 154.73 | 110.68 | 92.77 | 81.25 | 72.51 | 69.81 | 67.78 |
| FJS+ | 158.70 | 101.86 | 80.59 | 73.12 | 64.59 | 63.82 | 59.22 |
| HASH$q$ | 220.60$^{(2)}$ | 111.30$^{(2)}$ | 76.10$^{(2)}$ | 62.82$^{(3)}$ | 55.66$^{(5)}$ | 54.05$^{(5)}$ | 50.61$^{(4)}$ |
| FS-$w$ | 107.48$^{(6)}$ | 75.05$^{(6)}$ | 68.24$^{(6)}$ | 61.07$^{(6)}$ | 55.04$^{(8)}$ | 55.23$^{(6)}$ | 53.06$^{(8)}$ |
| LWFR$q$ | 89.08$^{(2)}$ | 74.73$^{(2)}$ | 59.29$^{(3)}$ | 53.79$^{(5)}$ | **50.58$^{(6)}$** | 49.78$^{(8)}$ | **48.80$^{(8)}$** |
| DIST$q$ | 89.64$^{(3)}$ | **67.32$^{(3)}$** | **58.45$^{(4)}$** | 54.43$^{(4)}$ | 51.15$^{(5)}$ | **49.72$^{(5)}$** | 49.02$^{(7)}$ |
| LDIST$q$ | 105.84$^{(2)}$ | 75.22$^{(3)}$ | 61.27$^{(4)}$ | 56.05$^{(5)}$ | 52.43$^{(5)}$ | 50.38$^{(5)}$ | 49.54$^{(5)}$ |

表 3．パターン出現数を調整した文字列

| $occ$ | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 | 131072 |
|---|---|---|---|---|---|---|---|
| BNDM$q$ | **55.64$^{(4)}$** | 58.24$^{(4)}$ | 61.02$^{(4)}$ | 65.78$^{(4)}$ | 76.84$^{(6)}$ | 97.77$^{(6)}$ | 121.10$^{(6)}$ |
| SBNDM$q$ | 56.03$^{(4)}$ | **56.75$^{(4)}$** | **59.57$^{(4)}$** | 67.30$^{(4)}$ | 73.93$^{(4)}$ | 93.81$^{(4)}$ | 121.00$^{(4)}$ |
| KBNDM | 80.05 | 83.24 | 86.06 | 92.19 | 110.03 | 129.52 | 161.81 |
| BSDM$q$ | 57.97$^{(4)}$ | 61.07$^{(4)}$ | 63.77$^{(4)}$ | 73.11$^{(6)}$ | 84.88$^{(6)}$ | 106.30$^{(6)}$ | 142.87$^{(6)}$ |
| FJS | 126.57 | 124.72 | 125.68 | 126.22 | 136.39 | 151.92 | 134.29 |
| FJS+ | 101.81 | 104.53 | 103.10 | 109.65 | 115.84 | 122.59 | 129.07 |
| HASH$q$ | 80.73$^{(3)}$ | 79.13$^{(4)}$ | 80.68$^{(4)}$ | 85.38$^{(4)}$ | 91.09$^{(4)}$ | 94.41$^{(4)}$ | 103.43$^{(5)}$ |
| FS-$w$ | 81.90$^{(6)}$ | 84.02$^{(6)}$ | 84.52$^{(4)}$ | 92.93$^{(4)}$ | 102.59$^{(6)}$ | 114.32$^{(4)}$ | 127.64$^{(2)}$ |
| LWFR$q$ | 57.92$^{(5)}$ | 61.59$^{(5)}$ | 68.48$^{(5)}$ | 78.38$^{(5)}$ | 97.94$^{(5)}$ | 131.67$^{(5)}$ | 185.21$^{(3)}$ |
| DIST$q$ | 57.96$^{(5)}$ | 57.62$^{(4)}$ | 60.67$^{(4)}$ | **63.42$^{(4)}$** | **69.44$^{(4)}$** | **80.87$^{(4)}$** | **98.21$^{(4)}$** |
| LDIST$q$ | 59.62$^{(4)}$ | 61.91$^{(5)}$ | 63.10$^{(4)}$ | 67.41$^{(5)}$ | 72.63$^{(4)}$ | 84.68$^{(4)}$ | 101.77$^{(4)}$ |

### 参考文献


[1] D. Cantone and S. Faro. Searching for a substring with constant extra-space complexity. In *Proceedings of Third International Conference on Fun with algorithms*, pp. 118–131, 2004.

[2] F. Franek, C.G. Jennings, and W.F. Smyth. A simple fast hybrid pattern-matching algorithm. *Journal of Discrete Algorithms*, Vol. 5, No. 4, pp. 682–695, 2007.

[3] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, Vol. 6, No. 2, pp. 323–350, 1977.

[4] D.M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, Vol. 33, No. 8, pp. 132–142, 1990.